IDN-12

Neighborly Protocols and Problems

Eric Rosen

November 1983

Here are some thoughts on the "how to choose a next-door neighbor" issue that we have been so actively discussing. There is some discussion of issues that we haven't really dealt with yet, viz., what the up/down protocol should be like, and what you have to do to become a next-door neighbor with someone else, once you have decided to do so. There is also a summary of my interpretation of what has been decided with respect to avoiding "virtual partitions"; this is a mixture of ideas from various sources. Finally, there is some discussion of what to do when, due to some careless operational mistake, we bring up two gateways with the same id number (the gateway-directed routing algorithm will not like this.)

If A and B are next-door neighbors, there is no need for A and B to be pinging each other, as well as replying to each

other's pings. Besides being wasteful, this raises the possibility that A will see B while B fails to see A. We want to avoid that situation. So perhaps we should use a protocol more like that of the ARPANET, in which the two neighbors have a master/slave relationship. The master just pings, and the slave just replies. The master decides when to declare the slave up, and informs the neighbor of its decision. The slave decides that the master is up when the master so informs it. The master can declare the slave down if it misses k out of n replies. The master should then cease pinging the slave for at least k intervals. The slave will declare the master down if it fails to receive any pings for at least k successive intervals. This procedure ought to ensure that whenever G1 regards G2 as a next-door neighbor, G2 also regards G1 as a next-door neighbor. At least, this will happen after a short transition period, which is no great problem.

Either side will declare the other down immediately if the directly connected network so indicates by means of that network's own access protocol (e.g., an 1822 dead is received). A gateway in its loader should reply to a ping with a negative response, indicating that it is in its loader; receiving such a

response will also cause the putative neighbor to be declared down. Whatever the event which causes one gateway to think a next door neighbor is down, it should force the neighbor to agree on the "down-ness" by refraining from sending pings or replies for awhile, as indicated in the previous paragraph.

If a next-door neighbor goes down, the gateway should not continue to ping it at the same rate as if it were up. It can be ignored totally for a certain time period (several minutes?), then pinged for a minute (or less, if an 1822 DEAD or some equivalent) is received, then ignored again, etc. Maybe the longer it is down, the longer the interval between attempting to bring it up again should be. Does this cause an unfortunate reduction in the responsiveness with which we detect that a gateway has come up? Not necessarily; when a gateway comes up, after all, it will try immediately to acquire some neighbors itself, rather than just waiting for some someone else to acquire it. Since a gateway always knows when it is coming up, different time constants can be used in this case. The lack of responsiveness might come when two neighboring gateways are both up, but out of communication with each other. We will discuss this more later. However, it seems clear that the current scheme

of continually (every 15 seconds) pinging the address of a gateway that has been gone for a month is too expensive.

In the ARPANET, the IMP with the greater of the two IMP numbers becomes the master, and the other becomes the slave. The same algorithm might as well be used in the gateway. This means that pings and their responses must carry gateway numbers, as well as addresses.

If a gateway G1 decides it wants to become a next door neighbor with a gateway G2, then some protocol is needed by which G1 can demand that G2 become its next door neighbor. A simple way to do this is just to have G1 ping G2. If G2 has a greater id number, it will respond by pinging G1. At this point, G1 must stop pinging G2, and start replying to its pings. If G2 has a smaller id number, it will just begin replying to G1's pings.

There is no reason why pings and responses should not be piggybacked on data packets, as long as we provide a rich enough internal header over the user-supplied IP leader.

The initial list of potential neighbor addresses will be a list of ADDRESSES, contained in the gateway's config area.

This list need not be comlete, but ought to be kept as complete and up-to-date as possible.The gateway will find out from pinging that address what the id number of the gateway there is. If the pings or replies suddenly indicate a different id number, something would appear to be wrong, and it might be a good idea to declare the neighbor down and try to re-acquire it.

(I am not as afraid of config area information as others may be, as long as the operational integrity of the system does not require that all changes in the config be made simultaneously in every gateway. With battery backed up semiconductor memory as our non-volatile memory, it is easy to change config data. The config area can have a checksum which is reported in the status messages, and compare on NU with what it should be. So it would be easy to know if the config information was somehow not properly updated, or if it was corrupted.)

There appears to be no need to have the gateway id numbers in the list of potential neighbors, or to have any kind of apriori mapping of numbers to addresses. A gateway with a given id number should be able to be moved from one host port to another, and then just come up, without causing any problems.

In general, we will want each gateway to have 2 next-door neighbors. (We certainly have to avoid the situation in which a given gateway has only 1 next-door neighbor, since that lacks robustness.) When a gateway comes up, it should start pinging a some of the addresses in its configured-in list of gateways, until it is able to acquire two next-door neighbors.

Once a gateway has a next-door neighbor or two, it will begin receiving routing updates. These updates will indicate the presence of additional gateways which are its neighbors (though not its next-door neighbors). We adopt the following convention:

> Let the neighboring gateways have numbers g[1],...,g[n], such that for all i, 1 <= i <= n, g[i] < g[i+1]. Then for all i, 2 <= i <= n-1, g[i] should be a next-door neighbor of g[i-1] and of g[i+1]. In addition, g[1] and g[n] should be next-door neighbors.

When a gateway first comes up, it does not necessarily pick the right set of next-door neighbors. But when it starts to send and receive routing updates, it learns the id numbers of everybody who is out there, and hence learns who should be its next-door neighbors.

Exactly how does a gateway G1 learn from a routing update

that it has a neighbor G2? If it receives a routing update whose source is G2, which indicates that it shares a network with G2, then it knows that G2 is a neighbor. Or it might receive a routing update whose source is G3, where G3 is a neighbor of G1, in which G3 indicates that G2 is one of its next-door neighbors on the net that G1 and G3 share. Of course, G1 should NOT assume that G2 is a neighbor unless this is a consequence of the transitive closure. (Let "n(G1, G2, N)" means "G1 and G2 are neighbors on net N". Then it is possible that n(G1, G2, N) & n(G1, G3, N) & ~n(G2, G3, N). This will happen, for example, if net N is partitioned, with G2 and G3 in different segments, but G1 is multihomed, and interfaces to an IMP in each segment. From transitivity, all we can derive if that if n(G1, G2, N) & n(G2, G3, N), then n(G1, G3, N).)

Suppose that a gateway has two next-door neighbors, but not the right two, according to our convention. Then it should acquire the "right two", in addition to the two it already has. However, this will cause it to end up with four next-door neighbors. So we need an algorithm to cause it to drop two of the neighbors. I suggest that this ability to drop neighbors be confined to the gateway on the "master" side of the connection.

It will drop a next-door neighbor whenever it has two "better" next-door neighbors. Presumably, the "slave" will also have two better next-door neighbors, and so will not attempt to re-acquire. Note, however, that it is certainly possible that the slave will just try to reacquire immediately (due to race conditions, perhaps), and that the protocol requires the master to become a next-door neighbor again, after which it will immediately try to drop again. However, if everybody agrees who is really out there and what id numbers they have, this situation should stabilize. If it doesn't, we have a problem, either in the transmission of routing updates, or in the transitivity assumption.

Another approach is to enhance the next-door neighbor protocol so that it takes prior agreement for two gateways to cease being next-door neighbors. Say, the master sets a bit indicating it wants to cease, and the slave either sets or clears a corresponding bit. I think I like this better; seems more robust.

If a gateway loses one of its favored (according to the convention) next-door neighbors, it should attempt to acquire the

next-best one. This means it has to look in its routing database to see which other neighbor (not a next-door one) should now become a next-door neighbor. If it fails to bring that one up, it should try the next-best, until it runs out of candidates.

Every gateway G1 should always expect to be a neighbor of any gateway in its initial list of potential neighbors. If some gateway G2 in that initial list is not reachable, then G1 should try periodically to become a next-door neighbor with it. This seems to mean that a dead gateway will be pinged by everyone else, from time to time. But I think that this is the only way to recover from partitions in the general case.

To prevent a gateway from sending an excess of pings to dead gateways, we can adopt the following strategy. The gateway can just ping one dead gateway at a time. If a negative response is received, it can just try the next gateway in the list. If a positive response is received, it can continue to try (for a fixed interval) to bring it up. If it fails, it moves on to the next gateway in the list. If no response at all is received to a single ping, it should try pinging the next gateway in the list. This would seem to maximize the chances of finding a gateway

which is actually up as soon as possible.

A gateway which is not in the configured list had by most other gateways can still join easily enough, as long as it knows of one other gateway. As soon as it becomes next-door neighbors with the one gateway it knows about, it gets routing updates, and can then join in choosing the proper pair of next-door neighbors according to our convention. But if this new gateway appears to the others to go down, should the other gateways "remember" it, and keep trying to re-acquire it, or should they forget it? The latter strategy has the advantage of making it easy to bring up temporary or test gateways without cluttering up the config area of the "core" gateways, or causing the core gateways to engage in excessive pinging of a gateway which is usually not going to be there. When we deploy additional core gateways, it is not so hard to alter the config area of the other gateways, so perhaps the approach of forgetting about gateways not in the config area makes the most sense.

In fact, in the case of test gateways, we might want to bring them up stand-alone, or in a little "test internet", without worrying that some core gateways are going to try to

acquire them as next-door neighbors. So we definitely don't want the core gateways to remember them.

We might want to further distinguish "test" gateways (running experimental software in the lab) from "temporary" gateways (like AERO, ROCKWELL, BRAGG, etc.). The latter are expected to be fully operational and robust while they are on-line, which should be for a period extending from several hours to several weeks. The former may crash at any time, and would often be running flaky software. Of course, while the software is real flaky (or while the routing software is suspect), test gateways should not become next-door neighbors of non-test gateways. But for some sorts of testing, we will want the test gateways to be neighbors of core gateways. However, we probably DON'T want any core gateway to have a test gateway as one (or both) of its two conventional next-door neighbors. Perhaps we ought to have a sequence of numbers (say 240-255) for test gateways, such that other gateways become next-door neighbors of them on demand only, and do not consider them when determining whether to drop or acquire any neighbor based on our convention.

Here is a routing-related problem which we haven't yet

discussed to any great degree.

One way to totally screw up a gateway-directed routing algorithm is to have two gateways with the same id number. The other gateways would have no way of knowing whether routing updates from gateway number G were from one or two gateways. This could result in mis-delivered packets and/or routing loops. Fortunately, the routing recovers automatically as soon as one of the two gateways goes down. The question is, how do we make a gateway go down (i.e., into its loader) if there is another gateway with the same id number?

The ARPANET contains two procedures to deal with this problem. When an IMP first comes up, there is a minute during which it receives routing updates, but does not send any. Furthermore, every IMP which is up is required to send a routing update at least every minute. If an IMP which is coming up receives an update during this minute which appears to be from itself, the IMP returns to its loader. This prevents an IMP from coming up with a number that is already in use.

However, this does not handle the problem that might occur if the network were partitioned, and each segment of the

partition contained an IMP with number N. Then when the partition ended, there would be two IMPs with the same number, and the above procedure would not help. What is done here is the following. If an IMP ever receives an update which appears to be from itself, but which mentions different neighbors than it actually has, the IMP falls into its loader. (Several years ago, the ARPANET and another net with IMPs numbered 1-6 were accidentally connected together through a test IMP which had a modem port on both networks. Fortunately, every IMP on the second net crashed, as well as IMPs 1-6 on the ARPANET.)

How do we know that the IMP is not just seeing one of its old updates, from a time when it had different neighbors. Well, it takes a minute to acquire a new neighbor, and there are procedures to make old updates die out after a minute, so that by the time an IMP has acquired a new neighbor, none of its updates mentioning old neighbors can still be floating around.

There is no pretense that these procedures make the network safe from sabotage. They are intended to protect it only from carelessness.

Unfortunately, this problem is harder to handle in the

internet environment, where we don't want to have to make use of one-minute time constants.

When a gateway is coming up, it gets routing updates forwarded to it from each of its neighbors. Suppose it gets an update that purports to be from itself. How does it know whether that update is a prior update from itself, or an update from another gateway with the same id number? Remember, the gateway's configuration could have changed radically in the meantime.

Maybe the following simple procedure is all we need. If a gateway receives an update which appears to be from itself, it should compare the sequence number of that update with the sequence number of the update it generated most recently. If the former sequence number seems to be "more recent" than the latter, or if the two sequence numbers are "out of range" (i.e., too far apart), something is wrong, and the gateway should crash into its loader. If there are two gateways with the same id number, this will cause at least one of them to crash, thereby restoring the integrity of the routing.

This will normally prevent a gateway from coming up with an id number which is already in use, since gateways will

remember, in their non-volatile memory, the last sequence number they used. Since a gateway will obtain routing information about the other gateways before beginning to send out its own routing updates, a gateway coming up with a number already in use should crash before it ever gets to send routing out at all. But what if some gateway has lost its non-volatile memory? This is actually the case in which we are most likely to experience trouble, since a gateway which has lost its non-volatile memory will not remember its id number, and its config area will have to be reloaded from the NOC. It wouldn't be too surprising to find that someone has loaded the wrong config information into the gateway, giving it the wrong gateway number.

Perhaps when a gateway is coming up, before it sends out any routing updates, it should make sure that no gateway with its same id number is reachable (according to routing). If the routing information obtained from its neighbors shows that a gateway with the same number is reachable, the gateway can crash into its loader.